Lancer de rayon : approches parallèles

Didier Badouel, François Bodin, Thierry Priol

INRIA

UNITE DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 992

Programme 2 / 5

LANCER DE RAYON : APPROCHES PARALLELES

Didier BADOUEL
François BODIN
Thierry PRIOL

Mars 1989

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Publication Interne n° 452 - Janvier 1989 - 16 Pages

Lancer de rayon : Approches parallèles

Didier BADOUEL
François BODIN
Thierry PRIOL

Résumé

Nous présentons deux approches pour la parallélisation de la méthode du lancer de rayon. La première est adaptée à l'utilisation d'une machine à mémoire distribuée. Nous présentons les résultats d'une mise en œuvre de ce type d'algorithme sur un hypercube iPSC/2. La deuxième méthode concerne l'utilisation d'un réseau linéaire de processeurs spécialisés pour accélérer les opérations géométriques du lancer de rayon.

Ray Tracing : Parallel Approaches

Abstract

We present two parallel approaches for the ray tracing technique. The first one is well suited for computers with distributed memory. The results of an implementation on a hypercube iPSC/2 are discussed. The second approach is based on the use of a network of dedicated processors to speed up the geometric operations during ray tracing synthesis.

Cet article correspond à une présentation effectuée lors des journées AFCET GROPLAN Langages et Algorithmes du Graphique qui se sont déroulées du 30 novembre au 2 décembre 1988 à Toulouse.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L. A. 227)
UNIVERSITÉ DE RENNES 1 I. N. S. A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

1 Présentation

La simulation des phénomènes photométriques à l'aide de la méthode du lancer de rayon a fortement contribué à améliorer la qualité des images de synthèse. Dans cet article, la présentation de la méthode du lancer de rayon est orientée vers une mise en œuvre sur des machines parallèles de cet algorithme très coûteux en temps de calcul. Une présentation succincte de la technique du lancer de rayon permet de mettre en évidence la part importante des opérations géométriques. Les optimisations algorithmiques qui sont exposées dans la deuxième partie sont la base des mises en œuvres sur des architectures parallèles présentées dans la dernière partie de cet article.

2 La technique du Lancer de rayon

Parmi les algorithmes de visualisation utilisés dans le domaine de la synthèse d'image, la méthode du lancer de rayon permet de créer des images avec un haut degré de réalisme. Cette méthode permet de simuler à la fois les phénomènes d'ombrage, de réflexion et de transparence. Son principe est tout à fait analogue à celui utilisé dans un appareil photographique.

Une analogie avec l'appareil photographique

L'idée développée par Goldstein et al. [GN71] était de simuler l'appareil photographique en suivant le trajet inverse de la lumière (cf Fig. 1). A l'intérieur de l'appareil photographique, la pellicule capte tous les rayons lumineux provenant de la scène photographiée. Avec la méthode du lancer de rayon, la pellicule est remplacée par un écran composé d'une matrice de pixels. Les rayons issus de l'observateur et passant par chacun des pixels de l'écran (rayons primaires), permettent de déterminer les objets visibles depuis la position de l'observateur. La couleur d'un pixel correspond à l'intensité lumineuse au point d'intersection entre le rayon primaire qui lui est associé et les objets de la scène. Cette intensité lumineuse correspond aux contributions directes et indirectes des sources lumineuses en ce point. A chaque point d'intersection entre un rayon et la scène, un nouveau rayon est lancé vers chaque source lumineuse (rayons lumineux) afin de connaître si le point est ombragé ou non. De même, de nouveaux rayons (rayons secondaires) lancés dans les directions transmises et r'efléchies permettent le rendu d'objets transparents

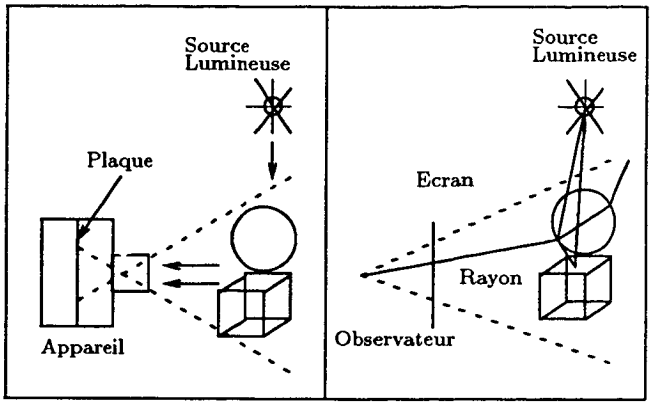

Figure 1: Analogie avec l'appareil photographique

ou réfléchissants. A partir de ces calculs et du modèle photométrique, on obtient l'intensité lumineuse des pixels.

Modélisation des objets

Un des choix qui conditionne l'algorithmique du lancer de rayon est le choix de la représentation des objets (modèle). Deux modélisations sont souvent employées:

1. Modélisation CSG.
2. Modélisation à l'aide de facettes planes.

La première consiste à utiliser des primitives volumiques simples (sphère, cylindre, parallélépipède,...) et à les composer à l'aide d'opérateurs binaires (union, différence, intersection). Cette modélisation est souvent employée pour la description d'objets mécaniques mais son utilisation apparaît limitée dans le cadre de la production audio-visuelle (modélisation de corps humains, de paysages, ...).

La représentation des objets à l'aide de facettes planes est celle que nous utilisons. Cette représentation est très souvent utilisée lors de la visualisation d'images de synthèse. Quelque soit le type de modélisation utilisé pour la création de scènes, il est toujours possible d'obtenir une approximation de la surface des objets à l'aide de polygones plans [BH88].

Le lancer de rayon : un algorithme coûteux en calcul

Dans cette représentation par facettes planes, environ 100 000 polygones peuvent être nécessaires pour décrire une image en production audio-visuelle. Le

calcul de cette image s'effectue avec une résolution de 512×512 pixels et correspond à la qualité d'un écran vidéo. Si en moyenne chaque pixel nécessite une dizaine de rayons, cela représente environ trois millions de rayons pour une image. A l'aide de cette *image de référence* (100 000 polygones et 3 millions de rayons) , le problème de la technique de lancer de rayon peut être quantifié de façon simple,

comment trouver 3 millions de fois l'intersection d'un rayon avec 100 000 polygones ?

Prenons comme première hypothèse, que tous les rayons sont comparés avec tous les polygones de la scène. Pour résoudre le problème des intersections, une image nécessite trois millions de fois le calcul de l'intersection entre un rayon et les 100000 polygones. Si la durée d'un calcul d'intersection rayon/polygone est d'environ 500 μs sur une machine du type SUN3, le temps total consacré aux intersections est de l'ordre de 4 ans !

C'est le nombre considérable d'intersections entre les facettes planes et les rayons qui induit un temps de calcul important. Les optimisations de l'algorithme du lancer de rayon portent donc sur l'ajout de structures permettant une diminution du nombre d'intersections. Ces structures peuvent être classées comme suit :

- **Les volumes englobants** : Cette technique associe à un objet ou ensemble d'objets un volume qui les englobe. L'intersection d'un rayon avec un volume englobant est moins coûteuse que l'intersection avec les objets. Si un rayon n'intersecte pas un volume englobant, alors aucun objet contenu dans celui-ci ne peut être intersecté. Ces volumes englobants, associés à une hiérarchie, contribuent fortement à diminuer le nombre d'intersections. Parmi ces volumes, nous pouvons citer les sphères englobantes de Whitted [Whi80], et les *Slabs* (Tranches) de Kajiya [Kaj83]. Les *Slabs* sont des domaines convexes qui permettent de décrire un volume englobant comme l'intersection de *Tranches* de l'espace.

- **Les structures régulières** : elles réalisent une subdivision uniforme de l'espace. Les objets peuvent être répartis dans une grille tridimensionnelle comme le font Fujimoto [FTI86], Cleary [CW88] et Amanatides [AW87]. Une approche complémentaire est celle de Ohta et Maekawa [OM87], qui ajoutent à la sudbivision régulière de l'espace une subdivision régulière des angles de vue. Le but de la méthode est de fournir une table de listes d'objets indexée par l'origine et la direction du rayon. Une subdivision adaptative peut être greffée autour de l'utilisation d'une grille tridimensionnelle. C'est le cas de la technique des macro-régions de Devillers [Dev88].

- **Les structures non régulières** : elles réalisent une subdivision adaptative de l'espace. Parmi celles-ci nous pouvons citer les techniques utilisant une partition binaire [APB87, Fuc80, Kap85], ou octale de l'espace [Gla84]. Ces méthodes minimisent un critère, par exemple le nombre d'objets contenus dans un sous-volume.

La comparaison objective des différents résultats des algorithmes de subdivision spatiale est difficile, car ils sont souvent expérimentés sur des machines différentes avec des scènes tests également différentes. Cependant, à la lecture des publications récentes, il apparaît clairement que les méthodes de subdivision régulière sont plus efficaces que les méthodes de subdivision adaptative pour le traitement de grosses scènes. Ceci est dû à la simplicité du parcours de la base de données.

3 Optimisations algorithmiques

Le propos de ce paragraphe est de présenter les algorithmes de base, pour les calculs géométriques, considérés dans notre étude. Nous ne présentons que les algorithmes concernant l'intersection d'un rayon avec la scène, sachant que cette partie représente plus de 80% du temps de calcul lors de la synthèse d'une image par lancer de rayon. Les algorithmes sont d'une part les *filtres* qui ont pour but de réduire le nombre d'intersections rayon/polygone nécessaires et d'autre part l'algorithme d'intersection entre un rayon et un polygone. Une description de chaque algorithme est reportée en annexe A.

Filtres

Un *filtre* est une procédure qui pour un rayon donné, reçoit une liste de polygones et effectue certains tests afin de réduire cette liste avant de la transmettre à la procédure suivante. Le premier filtre ($filtre_1$) opère sur l'ensemble des polygones de la scène. La liste provenant du dernier filtre ($filtre_n$) est traitée par la procédure calculant l'intersection effective entre un rayon et un polygone (*inter*). Avec ces filtres, l'expression du temps de calcul de l'intersection de R rayons avec l'ensemble de la scène est la suivante :

$$T = (T_{filtre_1}(P) + T_{filtre_2}(Nb_1) + \cdots + T_{filtre_n}(Nb_{n-1}) + T_{inter}(Nb_n)) \times R$$

- $T_{filtre_i}(n)$ correspond au temps de calcul du filtre i pour traiter n polygones.
- P correspond au nombre de polygones dans la scène.
- Nb_i correspond au nombre de polygones en sortie du filtre i.

Dans cette expression du temps de calcul, le prétraitement concernant l'initialisation des filtres n'est pas pris en compte. Il peut être considéré comme négligeable par rapport au volume de calcul lors de la synthèse d'une image, et a fortiori lors de la synthèse d'une séquence d'image. Nous souhaitons que l'ajout d'un filtre se traduise par une diminution du temps de calcul consacré aux intersections entre les rayons et la scène. Cela correspond à la propriété suivante :
Propriété :
si

$$T_0 = (T_{inter}(P)) \times R$$

$$T_i = (T_{filtre_1}(P) + \cdots + T_{filtre_i}(Nb_{i-1}) + T_{inter}(Nb_n)) \times R$$

alors

$$T_0 > T_1 > T_2 > \cdots > T_n \qquad (1)$$

La maquette logicielle sur laquelle nous travaillons actuellement est constituée de deux filtres. Le premier utilise une grille tridimensionnelle. Pour un rayon donné, la grille fournit une liste de polygones dont la localisation est proche de la direction du rayon. Ensuite, cette liste passe dans un second filtre qui élimine les polygones dont le volume englobant n'est pas rencontré par le rayon. Dans les annexes (cf. A), nous présentons l'utilisation d'une grille tridimensionnelle et de volumes englobants du type *Tranche d'espace*. Pour chacune des méthodes, nous essayons d'estimer les paramètres concernant les coûts en terme de calcul et de volume mémoire.

Utilisation conjointe des deux filtres

L'utilisation d'une grille tridimensionnelle impose un besoin important de mémoire (cf. formule (4) en annexe). A titre d'exemple, pour une scène contenant 100 000 polygones, la place mémoire utilisée par la grille peut être estimée à environ $6Mmots$.

Néanmoins, l'utilisation d'une structure de subdivision est indispensable car la complexité du temps de calcul est alors proche de $O(R)$. Dans le cas de l'utilisation d'une grille tridimensionnelle, la complexité est en $O(\sqrt[3]{P} \times R)$. Dans ce cas, la complexité est peu dépendante du nombre d'objets utilisés. Ce qui n'est pas vrai si on n'utilise que des volumes englobants où la complexité est en $O(P \times R)$.

L'utilisation conjointe de deux filtres (*grille* et *tranches d'espace*) permet d'utiliser une grille tridimensionnelle dont la résolution est plus faible. Dans ce cas, le nombre de voxels est moins important. De plus, les polygones occupent moins de voxels. A la lecture de la formule (4), on voit que la place mémoire occupée par la grille diminue de façon non négligeable. Ce gain compense aisément la place mémoire nécessaire pour mémoriser les informations concernant les *tranches d'espace* (cf. formule (6)). Le gain minimal obtenu est de l'ordre de deux.

L'utilisation conjointe de ces deux filtres respecte la propriété (1). Sur l'ensemble des images que nous avons testées, nous avons obtenu un gain en temps de calcul (pour les opérations géométriques) supérieur à 8%. Ceci s'explique par le fait que la grille qui minimise le temps de calcul (T) n'est pas celle qui filtre le plus de polygones.

Le principal intérêt des filtres est la diminution du temps de calcul. Une mise œuvre sur des architectures parallèles doit utiliser ce type de procédure comme traitement de base. Un deuxième intérêt est l'aspect modulaire de ces différentes procédures. Elles peuvent s'exécuter de façon relativement indépendante et utilisent des structures de données différentes. Ces caractéristiques peuvent être utilisées pour la parallélisation de l'algorithme.

4 Mise en œuvre sur des architectures parallèles

Nous présentons deux approches de la parallélisation du lancer de rayon. Nous excluons toutes les méthodes consistant à dupliquer les données sur chaque processeur. En effet, notre étude porte sur le traitement de scènes complexes représentant plusieurs dizaines de méga-octets de données qui ne peuvent être dupliquées dans chaque processeur.

La première approche consiste à paralléliser de façon globale l'algorithme de lancer de rayon et de l'implanter sur une architecture distribuée. Les calculs photométriques et géométriques sont distribués sur un ensemble de processeurs pouvant communiquer via un réseau d'interconnexion (hypercube, réseau maillé).

La deuxième approche parallélise uniquement les calculs géométriques car ce sont eux qui constituent l'essentiel des calculs. Elle correspond à l'approche proposée par Ullner [Ull83]. Un accélérateur de calcul, associé à un calculateur hôte, calcule les intersections entre les rayons et les facettes. Ce co-processeur

est lui-même composé d'un réseau linéaire de processeurs.

4.1 Une expérience sur une architecture à mémoire distribuée

Depuis peu, de nouvelles machines composées de plusieurs dizaines de processeurs sont disponibles (Ametek 2010, Intel iPSC/2, Meiko, Telmat T-node), offrant un rapport performance/coût plus intéressant que les super-ordinateurs. Toutes ces nouvelles architectures sont constituées d'un ensemble de processeurs disposant d'une mémoire locale et d'un réseau d'interconnexion permettant le transfert de messages entre processeurs. L'utilisation de ce type d'architecture nécessite un effort de programmation important. Notre étude porte sur une parallélisation d'un algorithme de lancer de rayon utilisant une technique de subdivision spatiale. Nous décrivons dans les paragraphes suivants cet algorithme ainsi que des résultats de son exécution sur un hypercube iPSC/2.

Partage équilibré des calculs

Afin de répartir à la fois les calculs et les données, nous avons choisi d'associer à chaque processeur un sous-volume, ou macro-région, obtenu par un découpage du volume englobant la scène [CWBV86]. Chaque processeur ne traite que les rayons entrant dans le volume qui lui est associé. Un des problèmes inhérent à cette technique est de déterminer la taille de chaque macro-région afin que chaque processeur ait la même charge de travail. Nemoto et al. [NO86] ont proposé une solution qui a pour principe de modifier en cours de calcul et aussi souvent que nécessaire la taille des macro-régions associées aux processeurs. Chaque processeur maintient une variable décrivant sa charge de travail, et connait également celle des processeurs voisins. Cette information est mise à jour périodiquement à l'aide de messages échangés entre processeurs voisins. Quand un processeur s'aperçoit qu'il a beaucoup plus de travail que ses voisins, il coopère avec eux pour redistribuer une partie de sa charge. Cette redistribution se traduit par une modification de la taille des macro-régions associées dans les processeurs concernés. Les objets contenus dans ces régions sont alors transférés aux processeurs correspondants. Cette technique est effectivement un bon moyen d'obtenir une charge de travail équilibrée au détriment toutefois de l'efficacité globale de l'algorithme. En effet, pour mettre en œuvre ce type de solution, les processeurs doivent échanger un nombre considérable d'informations, ce qui diminue d'autant l'efficacité de l'algorithme. Cette solution n'a pas fait

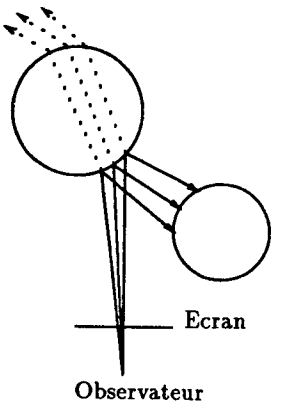

Figure 2: Propriété de cohérence des rayons

l'objet d'une mise en œuvre sur une machine parallèle, et il est donc difficile de l'évaluer.

Partitionnement a priori des données

Un des moyens pour calculer a priori la taille des macro-régions est d'utiliser la propriété de cohérence des rayons. Cette propriété a fait l'objet d'études ayant pour but de réduire les temps de calcul [HH84, Kaj82, SDB85, Whi80]. La cohérence est le fait que deux rayons passant par des pixels voisins ont une grande probabilité d'intersecter les mêmes objets. Comme le montre la figure 2, cette propriété se retrouve également dans l'évaluation des rayons lumineux, transmis et réfléchis.

De par la propriété de cohérence des rayons, le temps de traitement d'un pixel fournit une bonne estimation du temps de traitement des pixels voisins. Nous utilisons cette propriété afin de déterminer un partitionnement correct du volume de la scène [PB88]. La méthode de partitionnement que nous proposons est constituée de trois phases distinctes :

1. découpage du volume de la scène en cellules à l'aide d'une grille 2D;
2. sous-échantillonnage de l'image;
3. regroupement des cellules pour former des régions de complexité égale.

A l'issue de ces trois phases, l'espace de la scène est découpé en autant de régions que de processeurs disponibles dans la machine. Cette solution a été mis en œuvre sur un hypercube iPSC/2 et nous a permis de constater qu'un sous-échantillonnage correspondant à 1.5 % du nombre total de pixels dans l'image permet d'obtenir une charge de travail relativement bien équilibrée. En moyenne, chaque processeur calcule pendant 70 % du temps total de calcul de l'image.

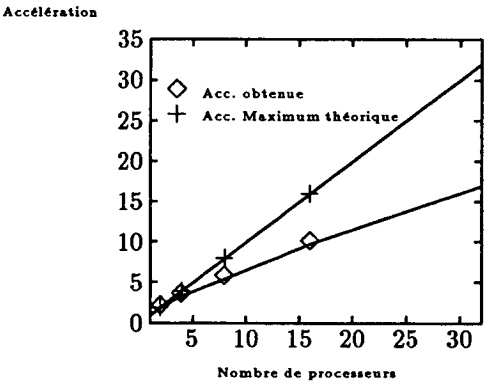

Figure 3: Accélération.

Un aperçu du fonctionnement de l'algorithme

Le fonctionnement de l'algorithme sur l'hypercube peut se résumer en trois étapes qui s'ajoutent au prétraitement décrit précédemment.

1. Le frontal envoie à chaque processeur une macro-région ainsi que la base de données correspondante. Il envoie également les informations de connexité de la macro-région. Au cours du traitement, lorsqu'un rayon quitte la macro-région associée au processeur, l'information de connexité permet de déterminer le processeur possédant la prochaine macro-région dans la direction du rayon.

2. Chaque processeur effectue une subdivision spatiale de la macro-région qu'il a reçu. A la fin de cette étape, un message est envoyé au processeur frontal pour lui indiquer que l'étape de subdivision est terminée. Quand le processeur frontal a reçu ce message de tous les processeurs, il autorise ceux-ci à débuter la phase de synthèse.

3. Cette dernière étape, ou tâche de synthèse, consiste à calculer les pixels de l'image. Chaque processeur s'occupe d'un sous-ensemble de pixels. Ce sous-ensemble dépend à la fois de la position et de la taille de la macro-région associée au processeur par rapport à l'écran. Il peut donc être vide, auquel cas le processeur a pour rôle de ne consommer que les rayons provenant des autres processeurs. Des files d'attente sont utilisées pour le stockage des rayons en attente de traitement, afin de ne pas bloquer le processeur émetteur.

Résultats

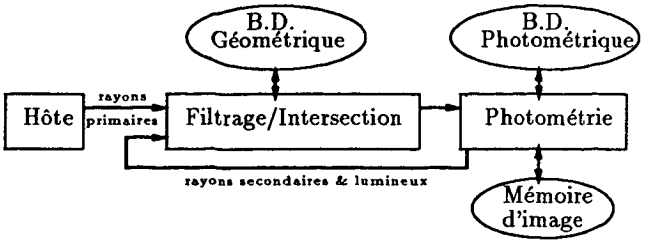

Figure 4: Organisation globale du lancer de rayon.

La figure 3 montre l'accélération en fonction du nombre de processeurs. On voit que l'accélération n'est pas linéaire par rapport au nombre de processeurs et que plus le nombre de processeurs augmente, plus l'efficacité de l'algorithme diminue. Ceci est dû à plusieurs raisons :

1. Le volume des communications augmente en fonction du nombre de processeurs. Ces communications entraînent des risques d'interblocage dûs à la saturation des files d'attente utilisées pour les transferts des rayons entre processeurs. Des techniques pour éviter l'interblocage sont donc nécessaires et elles contribuent à diminuer l'efficacité de l'algorithme.

2. Plus la scène est partitionnée en macro-régions, plus le volume de calcul augmente. En effet, à chaque fois qu'un rayon entre dans une nouvelle macro-région, il faut déterminer la cellule d'entrée de la grille.

En conclusion, ce type de parallélisation donne de bon résultats sur des architectures à mémoire distribuée, mais une parallélisation massive est limitée par non linéarité de l'accélération en fonction du nombre de processeurs.

4.2 Utilisation d'une architecture spécialisée

Avec la méthode de parallélisation présentée dans le paragraphe précédent le calcul des intersections et le calcul de la photométrie ne sont pas dissociés. Lorsqu'un processeur trouve une intersection dans le sous-volume de la scène qui lui est attribué, il évalue la contribution lumineuse correspondante. Cette méthode suppose donc un réseau d'interconnexion et nécessite un contrôle des communications relativement sophistiqué, afin d'éviter les problèmes inhérents aux calculs distribués, tels que l'interblocage, la détection de la terminaison.

Nous présentons une autre approche, de type *accélérateur de calcul*, avec un mode de communication plus simple, schématisé par la figure 4, dans laquelle le calcul géométrique et le calcul photométrique sont dissociées. Le module de calcul géométrique correspond à l'évaluation des intersections des rayons avec la scène. Le module photométrique gère la mémoire d'image et évalue les composantes du modèle photométrique. La valeur d'un pixel est évaluée par cumul des contributions de chaque rayon: en fonction des propriétés photométriques des objets intersectés, les rayons secondaires (rayons réfléchis, réfractés et lumineux) sont réinjectés dans le module géométrique afin d'évaluer la contribution des sources lumineuses, de la réflexion spéculaire et de la réfraction sur la surface de l'objet.

Nous nous intéressons ici au module géométrique qui nécessite la puissance de calcul la plus importante, en supposant que le calcul photométrique est assuré par la machine hôte. L'architecture cible pour le module géométrique est un réseau linéaire de processeurs PCS [1] [CR86]. PCS est un processeur cellulaire spécialisé pour le calcul rapide en arithmétique flottante, dont les domaines potentiels d'application sont la synthèse d'image, le traitement d'image, le traitement de signal, le calcul numérique. Les processeurs se connectent pour former un réseau linéaire. Les communications entre cellules sont effectuées par l'intermédiaire de deux files d'entrées de 512 mots, et supportent un débit de 80 Moctets/s. La réalisation d'un prototype de ce processeur est menée en collaboration avec la société SOGITEC avec le soutien de la Collaboration Bretagne Image (CBI). Le prototype est actuellement en phase de test, et sera intégré dans une station SUN3 début 1989.

Nous décrivons deux types de parallélisation que nous souhaitons évaluer sur un réseau de processeurs PCS. La première correspond à une approche *macro-pipeline* des opérations géométriques et la seconde à une *diffusion parallèle* des rayons.

Organisation des opérations géométriques en macro-pipeline

La présentation des algorithmiques (cf. paragraphe 3 et annexe A) fait ressortir un découpage possible des opérations géométriques en deux parties essentielles: le filtrage et le calcul d'intersection.

La figure 5 illustre la répartition des opérations géométriques sur les processeurs. Le macro-pipeline comporte trois étages. Le premier filtre P_1 utilise une grille tridimensionnelle. Pour un rayon donné, la

[1] Le processeur PCS est la suite de l'étude du processeur CSI menée dans le cadre d'une convention IRISA/CCETT (convention C9097X).

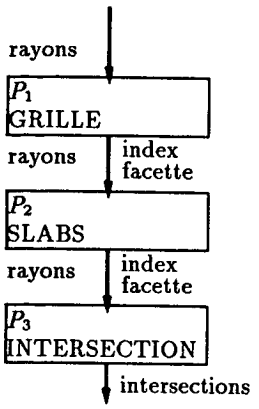

Figure 5: Macro-pipeline des opérations géométriques.

grille fournit une liste de polygones dont la localisation est proche de la direction du rayon. Ensuite cette liste passe dans un second filtre P_2 qui élimine les polygones dont le volume englobant n'est pas rencontré par le rayon. Le dernier étage du pipeline calcule l'intersection entre le plan de la facette et le rayon, et vérifie ensuite que l'intersection est à l'intérieur du polygone. Le polygone en sortie du pipeline est celui dont l'intersection est la plus proche de l'origine du rayon.

Les données sont réparties sur les processeurs. Le premier contient les listes d'index de facettes associées à chaque voxel de la grille. Sur la deuxième cellule on mémorise uniquement les données concernant les volumes englobants des facettes. La mémoire locale du dernier processeur contient les équations des plans des facettes, ainsi que la liste des sommets. La répartition des données réduit les transferts entre les processeurs puisque seuls les index des facettes et les rayons transitent entre les cellules. Les transferts entre l'hôte et le tableau de processeurs sont aussi réduits car l'hôte n'envoie que des rayons, et reçoit les intersections correspondantes en retour.

Pour que le macro-pipeline puisse être efficace, il faut que celui-ci soit constamment alimenté. Cela signifie d'une part que les processeurs doivent avoir une charge de travail équilibrée et d'autre part que les variations de flots de données entre chaque processeur n'entraînent pas une saturation des files d'attente. Les variations de flots de données les plus importantes se situent en sortie du processeur ayant en charge la gestion de la grille. La taille des FIFOs de PCS (deux fois 512 mots) permet de compenser ces variations de débit.

Cependant, cette organisation n'est définit que pour un petit nombre de processeurs. Ce caractère

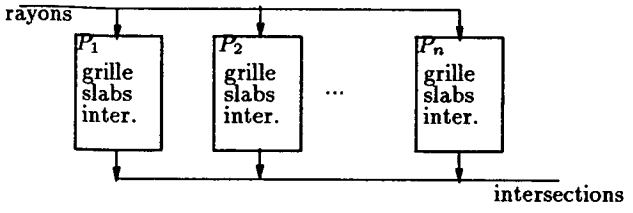

Figure 6: Diffusion parallèle des opérations géométriques.

non extensible du macro-pipeline nous conduit à étudier d'autres solutions.

Diffusion parallèle des rayons

Une autre technique de parallélisation est le découpage du volume de la scène en sous-volumes distribués aux processeurs. Les rayons sont diffusés à l'ensemble des processeurs, qui vont rechercher l'intersection avec les polygones qui leurs sont associés, de manière identique à l'algorithme séquentiel (cf. figure 6). Les résultats sont ensuite fusionnés, pour obtenir le premier polygone intersecté par le rayon.

Lorsqu'une cellule du réseau de processeurs reçoit un rayon, elle lui applique successivement les fonctions de filtrages (*Grille 3D*, *Tranches d'espace*) puis évalue l'intersection avec les polygones sortant du dernier filtre. Contrairement au cas précédent, chaque processeur possède toutes les informations concernant une partie seulement des polygones.

Pour mettre en œuvre cette parallélisation il faut pouvoir diffuser le rayon à tous les processeurs et ensuite fusionner les résultats. La diffusion des rayons ne peut être faite de manière directe, compte tenu des connexions entre les processeurs, mais en pipeline: chaque processeur transmet immédiatement à son voisin les rayons qu'il reçoit. Les communications entre processeur étant rapide, le surcoût imposé par ces transfert est négligeable devant le coût de calcul associé à un rayon. Les intersections calculées par les processeurs doivent être comparées afin d'extraire la première intersection. Pour ce faire chaque processeur compare son résultat avec celui de son voisin de gauche, et transmet à son voisin de droite, l'intersection la plus proche de l'origine du rayon.

Le surcoût en calcul engendré par cette parallélisation est le test d'intersection du rayon avec le volume associé au processeur et l'initialisation du parcours de grille. Ces calculs, effectués une seule fois dans l'algorithme séquentiel, sont réalisés sur chaque processeur et constituent la limite en temps de calcul que l'on puisse atteindre avec cette parallélisation.

	Grille	Slabs	Intersection
Initialisation/rayon	30 μs	9 μs	1 μs
Temps de calcul	7,8 μs /voxel	13 μs /poly.	30 μs /poly.

Figure 7: Simulation des opérations élémentaires

Le principal avantage de cette technique est que les communications par processeur n'augmentent pas en fonction du nombre de processeurs. Quelqu' en soit le nombre, le volume de communication traversant un processeur est $R \times (rayon + intersection)$. Cette remarque est très importante car, comme nous l'avons vu dans le §4.1, l'augmentation du volume de communication constitue une limite à une parallélisation massive de l'algorithme.

Cette technique de diffusion est actuellement en cours de simulation sur l'hypercube iPSC/2 configuré comme un réseau linéaire de processeurs.

5 Conclusion

Nous avons présenté plusieurs méthodes de parallélisation du lancer de rayon. La méthode qui consiste à distribuer le calcul sur une architecture générale à mémoire distribuée nous a permis d'obtenir des résultats intéressants. Elle fait ressortir les limites d'une parallélisation massive de ce type d'algorithme. Une parallélisation *idéale* devrait permettre d'obtenir une évolution linéaire du facteur d'accélération en fonction du nombre de processeurs tout en maintenant un rendement satisfaisant, c'est-à-dire un rapport entre le nombre de processeurs et l'accélération proche de un. Nous espérons obtenir un comportement intéressant de la méthode de *diffusion des rayons*. Cependant, si cette approche ne nous donne pas entière satisfaction, une approche plus pragmatique est son utilisation conjointe avec la méthode *macro-pipeline*.

A titre indicatif, le tableau de la figure 7 donne les performances obtenues pour les opérations géométriques élémentaires sur le processeur PCS.

Bibliographie

[APB87] B. Arnaldi, T. Priol, and K. Bouatouch. A new space subdivision for ray tracing CSG modelled scenes. *Visual Computer*, 3:98–108, 1987.

[AW87] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *EUROGRAPHICS'87*, pages 3–9, Amsterdam, 1987.

[BH88] D. Badouel and G. Hégron. An Evaluation of CSG Trees Based on Polyhedral Solids. In *EUROGRAPHICS'88*, Nice France, September 1988.

[CR86] F. Charot and F. Rousée. CSI: A Processor for Image Synthesis. In *EUROGRAPHICS'86*, Lisbonne, 1986.

[CW88] J.G. Cleary and G. Wyvill. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *The Visual Computer*, 4(2), July 1988.

[CWBV86] J.G. Cleary, B. Wyvill, G.M. Birtwistle, and R. Vatti. Multiprocessor ray tracing. *Computer Graphics Forum*, 5(1), March 1986.

[Dev88] O. Devillers. *Méthodes d'optimisation du tracé de rayons.* PhD thesis, Laboratoire informatique de l'ENS, rue d'Ulm, Paris, 1988.

[FTI86] A. Fujimoto, T. Tanaka, and K. Iwata. ARTS: accelerated ray_tracing system. *IEEE Computer graphics and applications*, April 1986.

[Fuc80] H. Fuchs. On visible surface generation by a priori tree structure. In *SIGGRAPH'80*, pages 149–158, July 1980.

[Gla84] A.S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer graphics and applications*, 4(10), 1984.

[GN71] R.A. Goldstein and R. Nagel. 3D visual simulation. *Simulation*, 16(1):25–31, January 1971.

[HH84] P. Heckbert and P. Hanrahan. Beam tracing polygonal objects. In *SIGGRAPH'84*, pages 119–129, 1984.

[Kaj82] J.T. Kajiya. Raytracing parametric patches. *IEEE Computer Graphics and Applications*, 16(3):245–254, March 1982.

[Kaj83] J.T. Kajiya. New techniques for ray tracing procedurally defined objects. In *SIGGRAPH'83*, pages 91–102, 1983.

[Kap85] M.R. Kaplan. Space-tracing, a constant time ray tracer. In *SIGGRAPH'85 tutorial on the uses of spatial coherence in ray tracing*, 1985.

[KK86] T.L. Kay and J.T. Kajiya. Ray tracing complex scenes. *ACM Computer Graphics*, 20(4), August 1986.

[Mar87] D.M. Marsh. *UgRay, An Efficient Ray-Tracing Renderer for Unigrafix.* Technical Report UCB/CSD 87/360, University of California, Berkeley, California, 1987.

[NO86] K. Nemoto and T. Omachi. An adaptative subdivision by sliding boundary surfaces for fast ray tracing. In *Graphics Interface'86*, pages 43–48, May 1986.

[OM87] M. Ohta and M. Maekawa. Ray coherence theorem and constant time ray tracing algorithm. In *Computer Graphics 1987 (Proceedings of CGI'87)*, pages 303–314, Tokyo, 1987.

[PB88] T. Priol and K. Bouatouch. Lancer de rayon sur des architectures parallèles: une étude de performance. In *Colloque C3*, Angoulème, December 1988.

[Pho75] B.T. Phong. Illumination for computer generated pictures. *Communication of the ACM*, 18(6):311–317, June 1975.

[SB87] J.M. Snyder and A.H. Barr. Ray tracing complex models containing surface tessellations. *ACM Computer Graphics*, 21(4), July 1987.

[SDB85] L.R. Speer, T.D. DeRose, and B.A. Barsky. A theorical and empirical analysis of coherent ray_tracing. *Graphics Interface'85*, May 1985.

[Ull83] M.K. Ullner. *Parallel machines for computer graphics.* PhD thesis, California Institute of Technology, 1983.

[Whi80] T. Whitted. An improved illmination model for shaded display. *Computer Graphics and Image Processing*, 23(6), June 1980.

A Algorithmes

Dans cette annexe, nous utiliserons les notations suivantes :

- t_x correspond au temps de calcul d'un traitement élémentaire x. Celui-ci dépend de la machine utilisée.

- nb_x correspond au nombre d'appels d'un traitement élémentaire x. Les évaluations de ces nb_x font appel à des techniques d'analyse statistique. Nous essayons actuellement de formaliser l'obtention de ces nombres à partir de quelques paramètres caractérisant une scène, tel que sa taille, le nombre de polygones, etc. La base de cette étude utilise les travaux de Cleary et al. [CW88] sur la technique des *volumes augmentés*, ainsi que les travaux d'analyse de Devillers [Dev88].

A.1 Grille tridimensionnelle

Une grille tridimensionnelle peut être vue comme une représentation discrète de l'espace scène, elle subdivise cet espace en sous-volumes parallélépipèdiques égaux appelés *voxels*. A chaque polygone est associé un *index* référençant les structures de données associées aux polygones. Chaque voxel de la grille possède une liste d'index des polygones coupant ce sous-volume.

Les polygones, pouvant être intersectés par le rayon, sont contenus dans les voxels rencontrés lors du cheminement du rayon dans la grille. La discrétisation du chemin parcouru par le rayon est *l'ensemble ordonné* de ces voxels.

Le parcours d'un rayon dans une grille tridimensionnelle, qui est un problème relativement similaire au problème du tracé de segment dans une grille bidimensionnelle, peut être résolu de deux manières différentes.

La première méthode apparue dans la littérature est celle utilisée par Fujimoto dans le système ARTS [FTI86]. Cette méthode utilise l'axe directeur du rayon. Cet axe correspond à la direction de plus grande pente, il est communément dénommé DA (Driving Axis). A chaque étape, l'algorithme se déplace d'une unité suivant cet axe. La progression dans une autre direction, i.e. suivant un axe passif, s'effectue suivant l'estimation de l'erreur commise sur cet axe. A chaque pas, celle-ci est incrémentée de la valeur constante représentant l'erreur commise suivant cet axe pour la traversée d'un voxel. Lorsque le seuil d'erreur (largeur du voxel) est dépassé, le parcours est incrémenté dans la direction de l'axe passif correspondant.

L'autre méthode, plus répandue, est utilisée par Amanatides & Woo [AW87], Snyder & Barr [SB87] et Cleary & Wyvill [CW88]. Cette méthode ne privilégie pas d'axe particulier. Une description détaillée de cette méthode est présentée dans [AW87]. Cependant, nous pouvons décrire succinctement cette méthode. Le rayon est représenté de façon paramétrique :

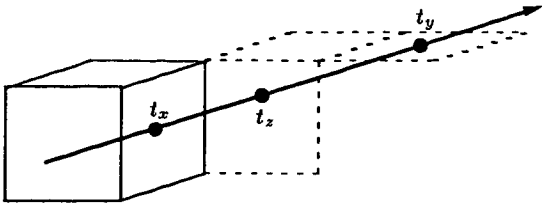

Figure 8: Parcours d'une grille tridimensionnelle

$$R(t) = O + D.t \tag{2}$$

- O est l'origine du rayon,
- D est le vecteur directeur du rayon, et
- t le paramètre de la représentation.

Initialement, le premier voxel parcouru par un rayon est calculé. Ce voxel est soit le voxel contenant l'origine du rayon, soit le voxel d'entrée dans la grille si l'origine du rayon est externe à la grille. Pour chaque rayon, les valeurs suivantes sont initialisées :

- δt_x, δt_y et δt_z: ces valeurs représentent les accroissements de t suivant chaque direction.
- t_x, t_y et t_z: ces valeurs représentent les valeurs de t lors du prochain franchissement d'une frontière de voxels pour chacune des directions (cf. Fig. 8).

Le parcours incrémental de la grille est réalisé alors de façon très simple. A chaque étape, la comparaison entre t_x, t_y et t_z indique dans quelle direction se trouve le prochain voxel à parcourir. A chaque pas dans la direction i, la variable t_i est incrémentée de la valeur δt_i. Le temps de calcul T des intersections entre les rayons et la scène, avec une grille tridimensionnelle, peut être exprimé comme suit :

$$T = (t_{init\,rayon\,grille} + nb_{pas} \times t_{pas} + nb_{filtre_1} \times t_{inter}) \times R \tag{3}$$

Le problème, commun à toutes les méthodes de subdivision spatiale, est de déterminer le meilleur niveau de subdivision pour une scène donnée. Un équilibre est à trouver entre le coût du filtre (parcours de la grille) et le coût du calcul utile (calcul d'intersection rayon/polygone), sachant que la grille qui minimise T n'est pas celle qui filtre le plus de polygones. D.M. Marsh

[Mar87] utilise dans le logiciel *UgRay* une heuristique basée sur des observations empiriques qui tend à montrer que le nombre de voxels (noté nb_{voxel}) doit être cinquante fois plus élevé que le nombre total de polygones dans la scène. Ce nombre (noté $coeff_{grille}$) définit la granularité de la grille.

$$nb_{voxel} = coeff_{grille} \times P$$

En ce qui concerne le nombre moyen d'itérations (nb_{pas}) effectuées par un rayon dans une grille, Devillers [Dev88] a montré que dans le cas d'une grille cubique, ce nombre peut être estimé au nombre de voxels suivant chaque direction. Nous avons observé que ce résultat peut être généralisé avec l'utilisation d'une grille non cubique en approchant le nombre moyen d'itérations par la racine cubique du nombre de voxels (nb_{voxel}). Cependant ce nombre correspond à la traversée complète de la grille. Une implémentation séquentielle peut tenir compte du fait que si une intersection est détectée dans un voxel donné, il est inutile de poursuivre le parcours du rayon dans la grille. Le résultat obtenu est un majorant de nb_{pas} :

$$nb_{pas} \leq \sqrt[3]{coeff_{grille} \times P}$$

Le dernier paramètre que nous devons évaluer est $nb_{poly/voxel}$, le nombre moyen de polygones par voxel. Ce taux d'occupation des voxels permet d'estimer le nombre de polygones en sortie d'une grille en fonction de sa granularité :

$$nb_{filtre_1} = nb_{pas} \times nb_{poly/voxel}$$

Le nombre moyen de polygones par voxel peut être évalué si l'on connaît $nb_{voxel/poly}$ le nombre de voxels occupés par un polygone :

$$nb_{poly/voxel} = \frac{nb_{voxel/poly} \times P}{nb_{voxel}} = \frac{nb_{voxel/poly}}{coeff_{grille}}$$

La technique des *volumes augmentés* (cf. [CW88]) nous fournit un instrument d'analyse bien adapté pour le calcul de $nb_{voxel/poly}$. Néanmoins, il reste quelques paramètres à évaluer, en particulier la taille moyenne des polygones pour une scène donnée. Existe-t'il une corrélation entre la taille d'une scène et le taille moyenne des polygones ? Sinon, la mesure des valeurs concernant la taille des polygones (périmètre et surface) doit être effectuée en prétraitement sur la scène.

Le volume mémoire occupé par une grille tridimensionnelle (mem_{grille}) peut être décomposé de la façon suivante :

$$mem_{grille} = mem_{table-3D} + mem_{listes\ d'index}$$

où $mem_{table-3D}$ correspond au nombre de voxels et $mem_{listes\ d'index}$ correspond au nombre de références aux polygones dans les voxels. On obtient :

$$mem_{grille} \simeq (coeff_{grille} + nb_{voxel/poly}) \times P \quad (4)$$

A.2 Les Tranches d'espace

Les Tranches d'espace ou *Slabs* (cf. [KK86]) sont des volumes englobants délimités par des paires de plans parallèles. Une tranche d'espace est caractérisée par une normale (notée N_i) et par deux valeurs d_i^{min} et d_i^{max}, telles que les équations des plans bordant le polygone dans la direction N_i, soient :

$$N_i \cdot P + d_i^{min} = 0 \quad et \quad N_i \cdot P + d_i^{max} = 0 \quad (5)$$

où P est un point du plan.

Les valeurs de d_i^{min} et d_i^{max} sont évaluées en projetant chaque sommet S_j du polygone sur la droite de direction N_i.

$$\begin{aligned} d_{ij} &= N_i \cdot S_j \\ d_i^{min} &= \min_j(d_{ij}) \\ d_i^{max} &= \max_j(d_{ij}) \end{aligned}$$

Les directions des tranches peuvent être différentes d'un polygone à l'autre. Cependant, afin d'effectuer des optimisations lors du calcul de l'intersection entre un rayon et un volume englobant, et afin de minimiser l'emplacement mémoire nécessaire, il est plus intéressant de choisir quelques directions privilégiées. Kay et Kajiya [KK86] proposent sept directions pour les *Slabs*.

$$N_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad N_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad N_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$N_4 = \begin{pmatrix} \frac{\sqrt{3}}{3} \\ \frac{\sqrt{3}}{3} \\ \frac{\sqrt{3}}{3} \end{pmatrix} \quad N_5 = \begin{pmatrix} \frac{-\sqrt{3}}{3} \\ \frac{\sqrt{3}}{3} \\ \frac{\sqrt{3}}{3} \end{pmatrix}$$

$$N_6 = \begin{pmatrix} \frac{-\sqrt{3}}{3} \\ \frac{-\sqrt{3}}{3} \\ \frac{\sqrt{3}}{3} \end{pmatrix} \quad N_7 = \begin{pmatrix} \frac{\sqrt{3}}{3} \\ \frac{-\sqrt{3}}{3} \\ \frac{\sqrt{3}}{3} \end{pmatrix}$$

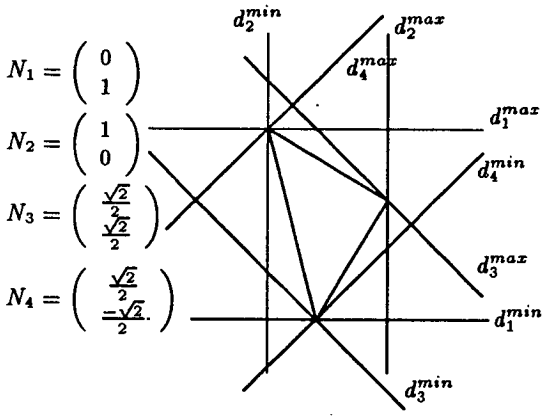

Figure 9: Exemple bidimensionnel de l'utilisation des 'Slabs' comme volume englobant.

Les valeurs d_i^{min} et d_i^{max} sont mémorisées pour chaque polygone. La place mémoire nécessaire est

$$2 \times n \; directions \times P \;\; mots \qquad (6)$$

Au cours du lancer de rayon, avant de calculer l'intersection entre un rayon et un polygone, l'intersection avec le volume englobant est évaluée. L'intersection d'un rayon avec les deux plans de chaque Slab nous fournit un segment de rayon compris entre t_i^{min} et t_i^{max}. Ces valeurs sont évaluées à l'aide de la représentation paramétrique du rayon (cf. Equ. (2)) et des équations des plans de chaque Slab (cf. Equ. (5)).

$$t_i^{min} = \frac{d_i^{min} - N_i.O}{N_i.D}$$
$$t_i^{max} = \frac{d_i^{max} - N_i.O}{N_i.D} \qquad (7)$$

Le fait de choisir des directions de 'Slabs' a priori, permet d'effectuer certaines optimisations. Lors de l'évaluation des valeurs de t_i^{min} et t_i^{max} (cf. Equ. (7)), seules les valeurs d_i^{min} et d_i^{max} dépendent du polygone testé. L'idée (proposée dans [KK86]) est donc de pré-calculer, lors du traitement d'un rayon, les valeurs

$$S_i = N_i \cdot O \quad et \quad T_i = \frac{1}{N_i \cdot D}$$

Ensuite, pour chaque intersection entre un rayon et une tranche d'espace, le calcul à effectuer est

$$t_i^{min} = (d_i^{min} - S_i)T_i \quad et \quad t_i^{max} = (d_i^{max} - S_i)T_i$$

L'intersection des intervalles $[t_i^{min}, t_i^{max}]$ est évaluée au fur et à mesure à l'aide de tests simples.

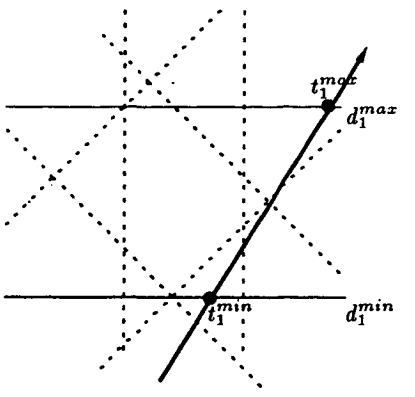

Figure 10: Calcul de l'intersection d'un rayon avec un volume englobant de type 'Slab'.

A partir du moment où l'intersection est vide, il n'est plus besoin de tester les tranches restantes. Une intersection non vide définit le segment de rayon $[t^{min}, t^{max}]$ interne au volume englobant du polygone, avec

$$t^{min} = \max_i(t_i^{min}) \quad et \quad t^{max} = \min_i(t_i^{max})$$

A.3 Intersection entre un rayon et une facette polygonale

Le but d'une procédure d'intersection entre un rayon et un polygone est de déterminer si le rayon traverse le polygone. De plus, lorsque l'intersection est effective, la procédure doit fournir les coordonnées du point d'intersection ainsi que des paramètres permettant de localiser le point dans le polygone. Ces paramètres permettent de calculer la normale au point d'intersection par interpolation des normales aux sommets, ce qui correspond à un lissage de type Phong [Pho75]. Ces paramètres permettent également le calcul des coordonnées dans une table de texture.

Intersection du plan support

Une facette polygonale est représentée par l'ensemble de ses sommets S_i (pour $i \in \{0, \cdots, N-1\}$). Le sommet S_i ayant comme coordonnées x_i, y_i et z_i. La normale au plan support de la facette, N, est évaluée à l'aide du produit vectoriel :

$$N = (S_1 - S_0) \times (S_2 - S_0)$$

Pour n'importe quel point P du plan on a $P.N = cste$. Cette valeur constante est calculée par le

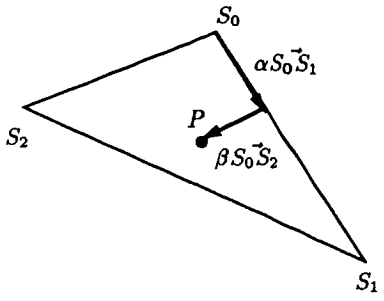

Figure 11: Représentation paramétrique du point P.

produit scalaire :

$$d = S_0.N$$

$$N.P + d = 0 \qquad (8)$$

Cette représentation arithmétique du plan est calculée une seule fois, puis stockée.

L'évaluation de l'intersection entre le rayon et le plan support du polygone consiste à calculer le paramètre t correspondant au point d'intersection, s'il existe, entre le rayon et le plan. A l'aide de l'équation (8) et de la représentation paramétrique du rayon (cf. équation (2)), on obtient

$$t = \frac{d - N.O}{N.D} \qquad (9)$$

Ce calcul nécessite douze opérations flottantes plus trois tests :

- Si le polygone et le rayon sont parallèles ($N.D = 0$), l'intersection est rejetée.
- Si l'intersection se situe avant l'origine du rayon ($t \leq 0$), l'intersection est rejetée.
- Si une intersection plus proche a déjà été trouvée ($t > min_t$), l'intersection est rejetée.

A partir du moment où le rayon coupe le plan support, il reste à déterminer si le point d'intersection se trouve à l'intérieur du polygone.

Intersection avec un polygone

Nous allons présenter la résolution du calcul d'intersection par une méthode vectorielle. La résolution est effectuée sur des polygones triangulaires. Le fait d'utiliser des triangles permet d'obtenir des algorithmes performants. Pour un polygone à n sommets ($n > 3$), celui-ci sera vu comme un ensemble de $n - 3$ triangles. La seule contrainte est l'utilisation de polygones convexes. Le point P (cf. Fig 5) est exprimé de la façon suivante :

$$\vec{S_0P} = \alpha.\vec{S_0S_1} + \beta.\vec{S_0S_2} \qquad (10)$$

Le point P sera interne au triangle $(S_0S_1S_2)$ si :

- $\alpha \geq 0$
- $\beta \geq 0$
- $\alpha + \beta \leq 1$

L'équation (10) revient à résoudre le système suivant :

$$\begin{cases} x_P - x_0 = \alpha(x_1 - x_0) + \beta(x_2 - x_0) \\ y_P - y_0 = \alpha(y_1 - y_0) + \beta(y_2 - y_0) \\ z_P - z_0 = \alpha(z_1 - z_0) + \beta(z_2 - z_0) \end{cases} \qquad (11)$$

Ce système de trois équations à deux inconnues peut se ramener à un système de deux équations à deux inconnues si l'on travaille dans un plan de projection. Le plan de projection sera l'un des trois plans perpandiculaires aux axes. Pour des raisons de précision de calcul et afin d'éliminer les cas dégénérés où la projection du triangle serait un segment de droite, on travaille dans le plan de plus grande projection. Pour cela, on calcule l'indice i_0 représentant la direction du plan de projection.

$$i_0 = \begin{cases} 0 & si\ |N_x| = Max(|N_x|, |N_y|, |N_z|). \\ 1 & si\ |N_y| = Max(|N_x|, |N_y|, |N_z|). \\ 2 & si\ |N_z| = Max(|N_x|, |N_y|, |N_z|). \end{cases}$$

Soit i_1 et i_2 (i_1 et $i_2 \in \{0, 1, 2\}$), les deux autres indices différents de i_0. Ces indices représentent les deux autres directions.

Après calcul des coordonnées bidimensionnelles (u, v) des vecteurs $\vec{S_0P}$, $\vec{S_0S_1}$ et $\vec{S_0S_2}$,

$$u_0 = P_{i_1} - S_{0_{i_1}}$$

$$v_0 = P_{i_2} - S_{0_{i_2}}$$

$$u_1 = S_{1_{i_1}} - S_{0_{i_1}}$$

$$v_1 = S_{1_{i_2}} - S_{0_{i_2}}$$

$$u_2 = S_{2_{i_1}} - S_{0_{i_1}}$$

$$v_2 = S_{2_{i_2}} - S_{0_{i_2}},$$

le système d'équations (11) devient :

$$\begin{cases} u_0 = \alpha.u_1 + \beta.u_2 \\ v_0 = \alpha.v_1 + \beta.v_2 \end{cases}$$

Les solutions du système sont :

$$\alpha = \frac{det\begin{pmatrix} u_0 & u_2 \\ v_0 & v_2 \end{pmatrix}}{det\begin{pmatrix} u_1 & u_2 \\ v_1 & v_2 \end{pmatrix}} \text{ et } \beta = \frac{det\begin{pmatrix} u_1 & u_0 \\ v_1 & v_0 \end{pmatrix}}{det\begin{pmatrix} u_1 & u_2 \\ v_1 & v_2 \end{pmatrix}}$$

Le calcul de α et β, une fois les coordonnées du point P évaluées, nécessite :

- 9 soustractions réelles;
- 6 multiplications réelles;
- 2 divisions réelles;

Lorsque le point P est interne au triangle, on évalue la normale à ce point à l'aide de l'interpolation :

$$N_P = (1 - (\alpha + \beta)).N_0 + \alpha.N_1 + \beta.N_2$$

LISTE DES DERNIERES PUBLICATIONS INTERNES

PI 447 **VISION 3D : UNE NOUVELLE METHODE DE LA TRIANGULATION POUR LA VISION MONOCULAIRE OU POLYNOCULAIRE DANS UNE SEQUENCE D'IMAGES**
Ming XIE, Patrick RIVES
48 Pages, Janvier 1989.

PI 448 **DYNAMIC VISION : DOES 3D SCENE PERCEPTION NECESSARILY NEED TWO CAMERAS OR JUST ONE ?**
Ming XIE
38 Pages, Janvier 1989.

PI 449 **CONVOLUTION SYSTOLIQUE DE FONCTIONS ARITHMETIQUES**
Patrice QUINTON, Yves ROBERT
30 Pages, Janvier 1989.

PI 450 **MISE EN OEUVRE D'ALGORITHMES NUMERIQUES SUR UN HYPERCUBE**
Brigitte VITAL
28 Pages, Janvier 1989.

PI 451 **LANCER DE RAYON SUR DES ARCHITECTURES PARALLELES : UNE ETUDE DE PERFORMANCE**
Thierry PRIOL, Kadi BOUATOUCH
20 Pages, Janvier 1989.

PI 452 **LANCER DE RAYON : APPROCHES PARALLELES**
Didier BADOUEL, François BODIN, Thierry PRIOL

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique